AAlign: A SIMD Framework for Pairwise Sequence Alignment on x86-based Multi- and Many-core Processors

Kaixi Hou, Hao Wang, Wu-chun Feng
{kaixihou, hwang121, wfeng}@vt.edu
Pairwise Sequence Alignment Algorithms

• Essential computational kernels in bioinformatics apps
  – Quantify similarity between pairs of sequences
• Different types of algorithms
  – Local alignment, e.g., Smith-Waterman
  – Global alignment, e.g., Needleman-Wunsch
• Different gap systems
  – Constant, linear, affine gap, etc.
Pairwise Sequence Alignment Algorithms

- Essential computational kernels in bioinformatics apps
  - Quantify similarity between pairs of sequences
- Different types of algorithms
  - Local alignment, e.g., Smith-Waterman
  - Global alignment, e.g., Needleman-Wunsch
- Different gap systems

These algorithms and gap systems ALL follow the same computational pattern.
Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods
Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

**Iterate**: use a certain number of iterations to validate results

---

1. Preprocess

Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

**Iterate**: use a certain number of iterations to validate results

1. Preprocess
2. Check


Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

**Iterate**: use a certain number of iterations to validate results

1. Preprocess
2. Check
3. Correct
Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

1. Preprocess
2. Check
3. Correct
Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

Iterate*: use a certain number of iterations to validate results

1. Preprocess
2. Check
3. Correct

Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

**Iterate**: use a certain number of iterations to validate results

**Scan**: use a round of scan to validate results

1. Preprocess
2. Check
3. Correct

Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

**Iterate**: use a certain number of iterations to validate results

**Scan**: use a round of scan to validate results

1. Preprocess
2. Check
3. Correct

---

Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

**Iterate**: use a certain number of iterations to validate results

**Scan**: use a round of scan to validate results

1. Preprocess
2. Check
3. Correct

1. Preprocess
2. Scan
Pairwise Sequence Alignment Algorithms

• Different Vectorization Strategies
  – Two popular strategies: iterate & scan methods

**Iterate**: use a certain number of iterations to validate results

**Scan**: use a round of scan to validate results

1. Preprocess
2. Check
3. Correct

---


Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

Iterate*: use a certain number of iterations to validate results

Scan**: use a round of scan to validate results

1. Preprocess
2. Check
3. Correct
Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

**Iterate**: use a certain number of iterations to validate results

**Scan**: use a round of scan to validate results

1. Preprocess
2. Check
3. Correct

1. Preprocess
2. Scan
3. Correct

Pairwise Sequence Alignment Algorithms

- Different Vectorization Strategies
  - Two popular strategies: iterate & scan methods

**Iterate**: use a certain number of iterations to validate results

**Scan**: use a round of scan to validate results

1. Need an unpredictable # of iterations to correct results
2. Do NOT need to visit all the values

1. Need a scan to correct results
2. Need to visit all the values
Motivation & Challenges

• Which vectorization strategy is better on x86 systems?
  – Affected by different algorithms, configurations, inputs & platforms

Different Algorithm with same affine gap over same input data

Different gap systems with same SW algorithm over same input data

Different inputs for the same SW algorithm with the same affine gap
Motivation & Challenges

• Which vectorization strategy is better on x86 systems?
  – Affected by different algorithms, configurations, inputs & platforms

1. Can we automatically generate the vector codes?
2. Can we design an even better vectorization strategy?

Different Algorithm with same affine gap over same input data

Different gap systems with same SW algorithm over same input data

Different inputs for the same SW algorithm with the same affine gap
Roadmap

- Introduction & Motivation
- Background
  - Vector ISA
- AAlign Framework
  - Generalized Paradigm
  - Vector Modules
  - Vector Code Constructs
  - Hybrid Method
- Evaluation & Discussion
- Conclusion
Background: Vector ISA

- Vector Processing Units
  - Carry out a single operation over a vector of data simultaneously

- AVX2 Instructions
  - **Platform:** Vector ISA in current multi-core Haswell CPUs
  - **Width:** 256 bits (two 128-bit lanes)
  - **Operations:** Gather, cross-lane permute, per-element shift, etc.

- IMCI Instructions
  - **Platform:** Vector ISA in many-core Knights Corner MIC
  - **Width:** 512 bits (four 128-bit lanes)
  - **Operations:** Scatter, gather, inner/cross-lane permute, etc.
Approaches to Using Vector ISA

- Compiler-based approaches
  - Compiler options
  - Pragma directives
Approaches to Using Vector ISA

- Compiler-based approaches
  - Compiler options
  -Pragma directives

**Issue:**
Fail to auto-vec loops due to complex memory access, convoluted data rearrangement, etc.
Approaches to Using Vector ISA

• Compiler-based approaches
  – Compiler options
  – Pragma directives

• Manual optimization via ...
  – Compiler intrinsics
  – Assembly code

Issue: Tedious and error-prone.
Approaches to Using Vector ISA

• Compiler-based approaches
  – Compiler options
  – Pragma directives

• Manual optimization via ...

Need a framework to hide the details of vector codes and make it cross-platform.

Serial C codes

```
for(i=0; i<2w; i++)
{
    if(i<offset)
        array[i]=x;
    else
        array[i]= array[i-offset];
}
```

Right-shift an array of length 2w

AVX2 intrinsics on CPUs

```
__m256i v_ret;
__m256i cv_rev = _mm256_set_epi32(6, 5, 4, 3, 2, 1, 0, 7);
v_ret = _mm256_load_si256((__m256i *)array);
v_ret = _mm256_permutevar8x32_epi32(v_ret, cv_rev);
v_ret = _mm256_insert_epi32(v_ret, x, 0);
```

IMCI intrinsics on MIC

```
__m512i v_ret;
__m512i cv_rev = _mm512_set_epi32(14,13,12,11,10,9,8,7,6,5,4,3,2,1,0,15);
unsigned short mask = 0xffff;     mask <<= num;
__m512i cv_fil = _mm512_set1_epi32(x);
v_ret = _mm512_load_epi32(array);
v_ret = _mm512_permutevar_epi32(cv_rev, v_ret);
v_ret = _mm512_mask_swizzle_epi32(cv_fil, mask, v_ret, _MM_SWIZ_REG_NONE);
```
Roadmap

• Introduction & Motivation
• Background
  – Vector ISA
• AAlign Framework
  – Generalized Paradigm
  – Vector Modules
  – Vector Code Constructs
  – Hybrid Method
• Evaluation & Discussion
• Conclusion
AAAlign Framework

- Architectural Overview
Proposed Generalized Paradigm

- Sequential codes follow our generalized paradigm
  - Support global and local alignment algorithms
  - Support different gap systems: constant, linear, affine

\[
T_{i,j} = \max\begin{cases}
0 \\
\max_{0 \leq l < j} (T_{i,l} + \theta_{i,l} + \sum_{k=l+1}^{j} \beta_{i,k}) \\
\max_{0 \leq l < i} (T_{l,j} + \theta'_{l,i} + \sum_{k=l+1}^{j} \beta_{k,j}) \\
T_{i-1,j-1} + \gamma_{i,j}
\end{cases}
\]

Example serial code follows the paradigm

1. for \( i \leftarrow 0; i < n+1; i++ \) do
2. \hspace{1em} \( T_{0,i} = U_{0,i} = L_{0,i} = 0 \);
3. for \( j < 0; j < m+1; j++ \) do
4. \hspace{1em} \( T_{j,0} = U_{j,0} = L_{j,0} = 0 \);
5. for \( i \leftarrow 1; i < n+1; i++ \) do
6. \hspace{1em} for \( j \leftarrow 1; j < m+1; j++ \) do
7. \hspace{2em} \( L_{i,j} = \max(L_{i-1,j} + \text{GAP}_\text{EX}, T_{i-1,j} + \text{GAP}_\text{OPEN}) \);
8. \hspace{2em} \( U_{i,j} = \max(U_{i,j} + \text{GAP}_\text{EXT}, T_{i,j-1} + \text{GAP}_\text{OPEN}) \);
9. \hspace{2em} \( D_{i,j} = T_{i-1,j-1} + \text{BLOSUM62}_{\text{ctoi}(Q_{i-1}), \text{ctoi}(S_{j-1})} \);
10. \hspace{2em} \( T_{i,j} = \max(0, L_{i,j}, U_{i,j}, D_{i,j}) \);
11. // resultant score is the maximum value in \( T \)
Vector Operation Modules

- Used to express required primitive vector operations
- Basic vector operations (e.g., load/store/add/max)
- Application-specific vector operations

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_vector</td>
<td>Initialize a new vector using the given gap info.</td>
</tr>
<tr>
<td>rshift_x_fill</td>
<td>Right shift the vector and fill the gaps with specified value $x$</td>
</tr>
<tr>
<td>influence_test</td>
<td>Check if vector can affect another vector</td>
</tr>
<tr>
<td>wgt_max_scan</td>
<td>“weighted” max-scan over the values in a given array using vectorized method</td>
</tr>
</tbody>
</table>
Portability of Vector Modules

- **Example:** `rshift_x_fill()`
### $rshift_x_fill$ (IMCI 32-bit int)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

16 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

x x x x x x x x x x x x x

x 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

- **__m512_permutevar_epi32**
- **__m512_set1_epi32**
- **__m512_mask_swizzle_epi32**

### $rshift_x_fill$ (AVX2 32-bit int)

1 2 3 4 5 6 7 8

8 1 2 3 4 5 6 7

x 1 2 3 4 5 6 7

- **__m256_permutevar_epi32**
- **__m256_insert_epi32**
Portability of Vector Modules

• **Example:** `rshift_x_fill()`

```
rshift_x_fill (AVX2 32-bit int)

1 2 3 4 5 6 7 8
8 1 2 3 4 5 6 7
x 1 2 3 4 5 6 7

__m256_permutevar_epi32
__m256_insert_epi32

```

```
rshift_x_fill (AVX2 16-bit int)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
4 1 2 3 8 5 6 7 12 9 10 11 16 13 14 15
16 13 14 15 4 1 2 3 8 5 6 7 12 9 10 11
16 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
x 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

__m256_shufflehi/lo_epi16
__m256_permutevar8x32_epi16
__m256_blend_epi16
__m256_insert_epi16
```
Portability of Vector Modules

- **Example:** \textit{rshift\_x\_fill()}

Provide such portability of the same functionality over different ISAs and built-in datatypes.
Vector Code Constructs

- Striped layout of the query sequence
  - SIMD-friendly due to the elimination of data-dependency among adjacent elements

Original:  

\[
\begin{array}{cccccc}
  & a & b & c & d & e \\
  i & A & B & C & D & E \\
\end{array}
\]

Striped:  

\[
\begin{array}{cccccc}
  & a & A & b & B & c & C & d & D & e & E \\
  i & & & & & & & & & &
\end{array}
\]

- Both “iterate” and “scan” methods can use the striped layout
- Provide foundations of merging the two methods
Vector Code Constructs

• Striped-Iterate
  – Iteratively correct the results if the updated predefined vectors affect the results

• Striped-Scan
  – Correct the results using the vectorized “weighted” scan

---

**Function aalign_iterate()**

```c
// Preprocess the column values

int j = 0;
REC_UP = rshift_x_fill(REC_UP, 1, REC_UP);

while (influence_test(REC_UP, REC_UP, REC_UP)) {
    vT = load_vector(arrT1 + j * vec_len);
    vT = max_vector(vT, REC_UP);
    store_vector(arrT2 + j * vec_len, vT);
    REC_UP = add_vector(REC_UP, REC_UP);
    if ++j >= k then
        REC_UP = rshift_x_fill(REC_UP, 1, REC_UP);
    }

swap(arrT1, arrT2);
```

**Function aalign_scan()**

```c
// Preprocess the column values

wgt_max_scan(arrT2, arrScan, m, INIT_T, GAP_UP_EXT, GAP_UP)

for j ← 0; j < k; j++ do
    vT = load_vector(arrScan + j * vec_len);
    vT = load_vector(arrT2 + j * vec_len);
    vT = max_vector(vT, vT);
    vT = max_vector(vT, vT);
    vT = store_vector(arrT2 + j * vec_len, vT);
    swap(arrT1, arrT2);
```
Hybrid Method

- Can we design an even better method?
- Observations (e.g., SW with affine gap)
Hybrid Method

• Can we design an even better method?
• Observations (e.g., SW with affine gap)

If two sequences are similar, "iterate" checks each position with more re-computation steps to eliminate false negatives
Hybrid Method

- Can we design an even better method?
- Observations (e.g., SW with affine gap)

If two sequences are not similar, only a few rounds of re-computation steps are needed.
Hybrid Method

• Can we design an even better method?
• Observations (e.g., SW with affine gap)

In "scan", the re-computations steps are fixed, i.e. scan, correction.
Hybrid Method

- Our idea: Automatically switch to the better strategy based on the current # of re-computations at runtime
Hybrid Method

- Our idea: Automatically switch to the better strategy based on the current # of re-computations at runtime
Hybrid Method

• Our idea: Automatically switch to the better strategy based on the current # of re-computations at runtime

1. Switch

2. "Probe": noop
Hybrid Method

- Our idea: Automatically switch to the better strategy based on the current # of re-computations at runtime
Hybrid Method

- Our idea: Automatically switch to the better strategy based on the current # of re-computations at runtime
Details of the AAlign Framework

• Code translation
  – Use clang driver to create Abstract Syntax Tree (AST)
  – Detect the type of sequential code (e.g., SW or NW; linear or affine gap; etc.)
  – Create real vector codes based on vector code constructs and type information

• Multi-threaded version
  – Perform one-to-all sequence alignment
  – Database sequences have been sorted for better load balance
Roadmap

• Introduction & Motivation
• Background
  – Vector ISA
• AAlign Framework
  – Generalized Paradigm
  – Vector Modules
  – Vector Code Constructs
  – Hybrid Method
• Evaluation & Discussion
• Conclusion
Evaluation & Discussion

• Experiment Setup

– The queries are real sequences selected from NCBI, i.e.,
  AL4A1_HUMAN(Q500), COSA1_HUMAN(Q1000), B0I1R8_HUMAN(Q2000), MUC17_HUMAN(Q4000)

– The database is “Swiss-prot” containing over 570k sequences

<table>
<thead>
<tr>
<th>Parameter</th>
<th>CPU</th>
<th>MIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Name</td>
<td>Intel Xeon E5-2680 v3</td>
<td>Intel Xeon Phi 5110P</td>
</tr>
<tr>
<td>Code Name</td>
<td>Haswell</td>
<td>Knights Corner</td>
</tr>
<tr>
<td># of Cores</td>
<td>24</td>
<td>60</td>
</tr>
<tr>
<td>Clock Rate</td>
<td>2.5 GHz</td>
<td>1.05 GHz</td>
</tr>
<tr>
<td>L1/L2/L3 Cache</td>
<td>32 KB/ 256 KB/ 30 MB</td>
<td>32 KB/ 512 KB/ -</td>
</tr>
<tr>
<td>Memory</td>
<td>128 GB DDR3</td>
<td>8 GB GDDR5</td>
</tr>
<tr>
<td>Compiler</td>
<td>icpc 15.3</td>
<td>icpc 15.3</td>
</tr>
<tr>
<td>Compiler Options</td>
<td>-xCORE-AVX2 –O3</td>
<td>-mmic -O3</td>
</tr>
<tr>
<td>Vector ISA</td>
<td>AVX2</td>
<td>IMCI</td>
</tr>
</tbody>
</table>
Evaluation & Discussion

- Speedups over Serial Counterparts

  - “AAlign-Iterate” can achieve better vectorization efficiency
  - Superlinear speedups for “AAlign-Iterate” comes from the elimination of a considerable amount of re-computation when the `influence_test()` fails
Evaluation & Discussion

- Performance of Hybrid Method

- Use query coverage (QC) and max identity (MI) to describe the similarity of two sequences. (Format: <QC>_<MI>, e.g., lo_hi)
Evaluation & Discussion

• Performance of Hybrid Method

- Use query coverage (QC) and max identity (MI) to describe the similarity of two sequences. (Format: \(<QC>_<MI>\), e.g., lo_hi)

*lo_hi* means only a small portion (<30%) of two sequences overlaps each other, but the overlapped areas are highly similar (>70%).

**SW on Haswell CPU**

**SW on Knights Corner MIC**
Evaluation & Discussion

- **Performance of Hybrid Method**

  - Use query coverage (QC) and max identity (MI) to describe the similarity of two sequences. (Format: <QC>_<MI>, e.g., lo_hi)
  
  - Hybrid method can achieve better performance than both vector algorithms; for some cases, it can approximate the superior one.

---

*SW on Haswell CPU*

*SW on Knights Corner MIC*
Performance Comparison with Open-Source Tools

- **AAlign vs. SWPS3** on CPU
  - CPU: AAlign codes can outperform SWPS3 by up to 2.5x
- **AAlign vs. SWAPHI** on MIC
  - MIC: AAlign codes can outperform SWAPHI by up to 1.6x

---


Conclusion

• AAlign: A specialized framework for pairwise alignment algorithms on the x86-based processors
  – Efficient vector codes based on “striped-iterate” & “striped-scan”
  – Sets of platform-specific vector modules

• Design: A new input-agnostic hybrid method

• Performance:
  – Significant performance gains over serial counterparts
  – Auto-switching to better vectorization strategy at runtime (hybrid method)
  – Up to 2.5x performance benefit over existing multi-threaded tools

• Availability: https://github.com/vtsynergy/aalign

THANK YOU!

More info: http://synergy.cs.vt.edu